Corso di Object Oriented Software Design
a.a. 2018/2019

Documento di design

Membri del team		
Nome	**Matricola**	**Indirizzo E-mail**
Salvatore Salernitano	*242016*	*salvatore.salernitano@student.univaq.it*
Lorenzo Salvi	*242387*	*lorenzo.salvi@student.univaq.it*
Ludovico Di Federico	*243542*	*ludovico.difederico@student.univaq.it*

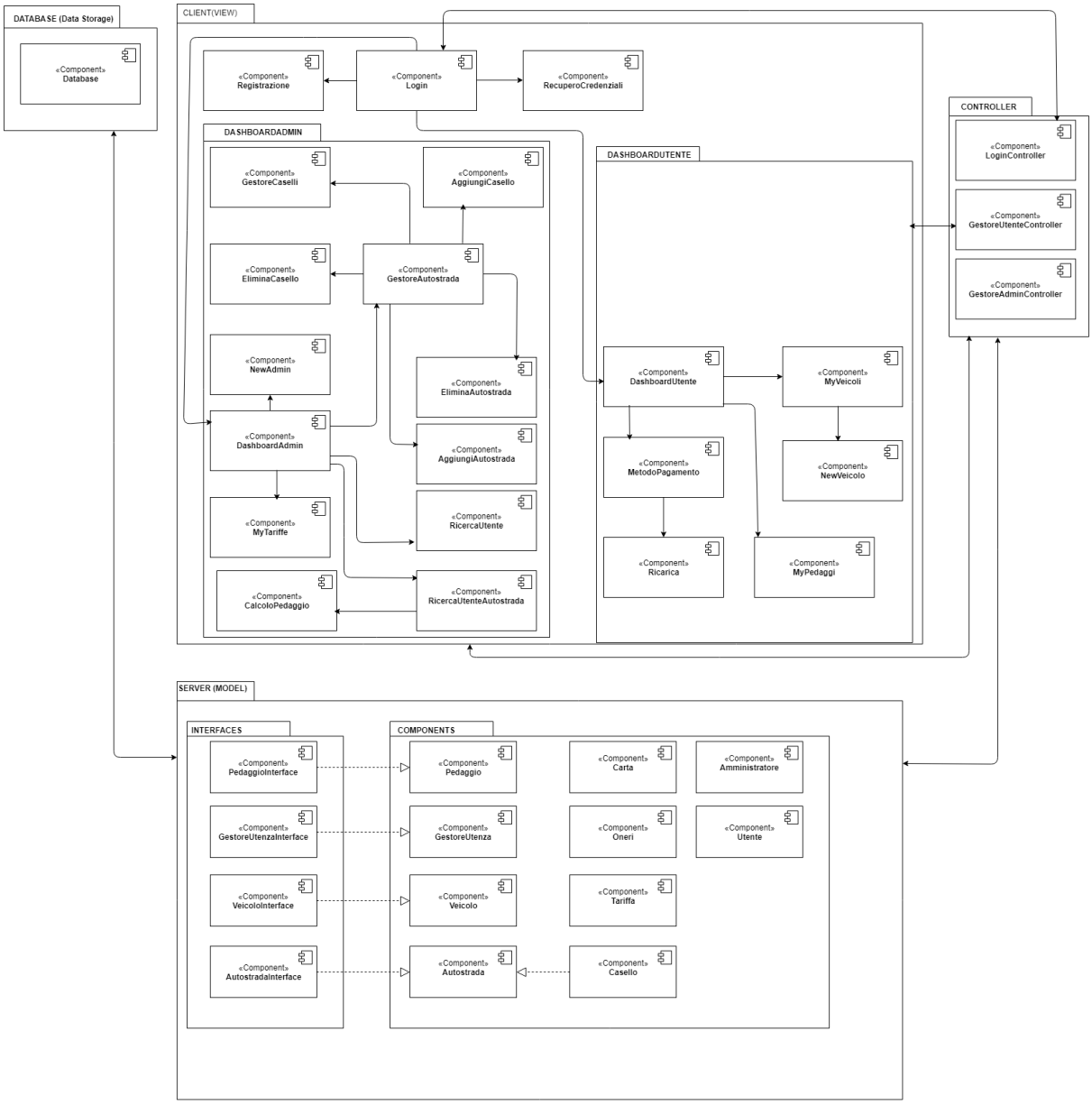
a. REQUISITI

(1.1 REQUISITI FUNZIONALI):

- L'**amministratore** del sistema può aggiungere, modificare ed eliminare un'autostrada con i rispettivi caselli;
- L'**amministratore** del sistema può gestire le tariffe unitarie tenendo conto delle categorie dei veicoli (A, B, 3, 4, 5), della tipologia di autostrada (montagna o pianura) e dei relativi oneri per i costi esterni (in base alla quantità di CO2 emessa dal veicolo);
- L'**amministratore** del sistema può visualizzare le informazioni relative ad un determinato utente ed i veicoli d'appartenenza;
- L'**amministratore** del sistema potrà effettuare il calcolo del pedaggio stradale utilizzando le informazioni relative al percorso ed alla targa del veicolo dell'utente;
- L'**utente** può aggiungere ed eliminare uno o più veicoli;
- L'**utente** potrà **visualizzare** i pedaggi pagati, se il pedaggio risulterà "non pagato" l'utente **procederà al pagamento** mediante contante o carta. Nel caso in cui l'utente, nel momento del pagamento, risulti avere il saldo della carta inferiore al costo del pedaggio, l'utente **dovrà effettuare una ricarica**;
- L'**amministratore** del sistema può aggiungere un nuovo amministratore all'interno del sistema;
- L'**utente** può registrarsi all'interno del sistema;
- L'**amministratore** e l'**utente** possono **accedere al sistema** mediante le credenziali d'accesso (username e password) e **recuperare le credenziali** mediante una chiave di recupero (inserita da essi durante la registrazione).

b. ARCHITETTURA DEL SOFTWARE

(2.1 Modello dell'Architettura Software – Diagramma con Packages)



(2.2 Descrizione dell'Architettura)

Il team ha deciso di utilizzare l'MVC pattern (Model-View-Controller), ovvero un pattern architetturale molto diffuso nello sviluppo Software, in particolare nell'ambito della programmazione Object-Oriented.

Il pattern è basato sulla separazione dei compiti fra le componenti software che interpretano tre ruoli principali:

- **Model** fornisce metodi per accedere ai dati utili all'applicazione;
- **View** visualizza i dati contenuti nel model e si occupa dell'interazione tra utenti ed agenti;
- **Controller** riceve i comandi dell'utente, in genere attraverso una View, e li attua modificando lo stato degli altri due componenti.

Le macro componenti che suddividono la nostra architettura, sono le seguenti:

- **Database (Data Storage):** il sistema dovrà includere *"database"*, ovvero una component che ha il compito di immagazzinare i dati;
- **Client (GUI):** conterrà due *packeges* chiamati rispettivamente *"DashboardAdmin"* e *"DashboardUtente"*, che al loro interno avranno come componenti: *"DashboardAdmin"*, "*NewAdmin*", *"MyTariffe"*, *"CalcoloPedaggio"*, *"GestoreAutostrada"*, *"GestoreCaselli"*, *"EliminaCasello"*, *"AggiungiCasello"*, *"EliminaAutostrada"*, *"AggiungiAutostrada"*, *"RicercaUtenteAutostrada"*,*"RicercaUtente"*, *"DashboardUtente"*, *"MyVeicoli"*, *"NewVeicolo"*, *"MetodoPagamento"*, *"MyPedaggi"* e *"Ricarica"*. Questo macro blocco ha il compito di interfacciare l'utente con il sistema (View). *"DashboardAdmin"* rappresenta l'interfaccia grafica tra l'Amministratore del sistema ed il sistema stesso, mentre la *"DashboardUtente"* rappresenta l'interfaccia grafica tra l'Utente/Cliente ed il sistema di gestione dei pedaggi autostradali. Il component "*Login*" svolge un ruolo importante, ovvero ha il compito di identificare un determinato utilizzatore del sistema che effettua l'accesso in modo tale da garantire la protezione da attacchi esterni. Il component "*Registrazione*" ha il compito di registrare un nuovo utente all'interno del sistema, mentre il component *"RecuperoCredenziali"* permette il recupero delle credenziali d'accesso (Username e Password) da parte di un Utente o un Amministratore.
- **Controller:** è una macro componente che ha il compito di elaborare le richieste in ingresso, gestire gli input e le interazioni del singolo utente o amministratore per eseguire la logica del sistema appropriato. Al suo interno troveremo tre componenti denominate rispettivamente: *"LoginController"*, *"GestoreUtenteController"*, *"GestoreAdminController"*;
- **Model (Server):** è una macro componente che fornisce i metodi per accedere ai dati utili all'applicazione (Model). Il model non si occupa soltanto dell'accesso fisico ai dati, ma anche di creare il necessario livello di astrazione tra il formato in cui i dati sono memorizzati ed il formato in cui i livelli di Controller e View si aspettano di riceverli. Oltretutto fornisce un'interpretazione intermedia dei dati arricchendo il database con nuove informazioni. Cosa importante, il Model non contiene direttamente i dati del Database, ma ha solo il compito di fornire i metodi e di restituire i dati al Controller. La macro componente Server conterrà due *packages* chiamati rispettivamente *"Interfaces*" e *"Components"*. Le components presenti all'interno del package *"Components"* sono: *"Pedaggio"*, ovvero una classe che implementa *"PedaggioInterface*" e conterrà tutti i metodi inerenti al calcolo del

pedaggio e alle impostazioni inerenti alle tariffe e agli oneri; *"GestoreUtenza"*, ovvero una classe che implementa *"GestoreUtenzaInterface"* che conterrà tutti i metodi inerenti alla gestione dell'utenza (quali accesso, restituzione delle credenziali d'accesso e registrazione); "*Veicolo*" ovvero una classe che implementa *"VeicoloInterface"* che conterrà tutti i metodi inerenti alla gestione dei veicoli; *"Autostrada*", ovvero una classe che implementa *"AutostradaInterface"* che conterrà tutti i metodi inerenti alla gestione dell'autostrada e dei caselli. Questa componente dipenderà dalla componente *"Casello"* che gestirà tutte le informazioni dell'autostrada; *"Carta"*, ovvero una classe che conterrà tutte le informazioni inerenti al metodo di pagamento tramite carta; *"Tariffa*", ovvero una classe che conterrà tutte le informazioni inerenti alle tariffe applicate sui veicoli e sulle autostrade; una component chiamata *"Oneri",* ovvero una classe che conterrà tutte le informazioni inerenti agli oneri per costi esterni applicati a seconda delle emissioni di CO2; una component chiamata "*Amministratore*", ovvero una classe che conterrà tutte le informazioni inerenti ad un Amministratore; una component chiamata "*Utente*" che a sua volta conterrà tutte le informazioni inerenti all'utente.

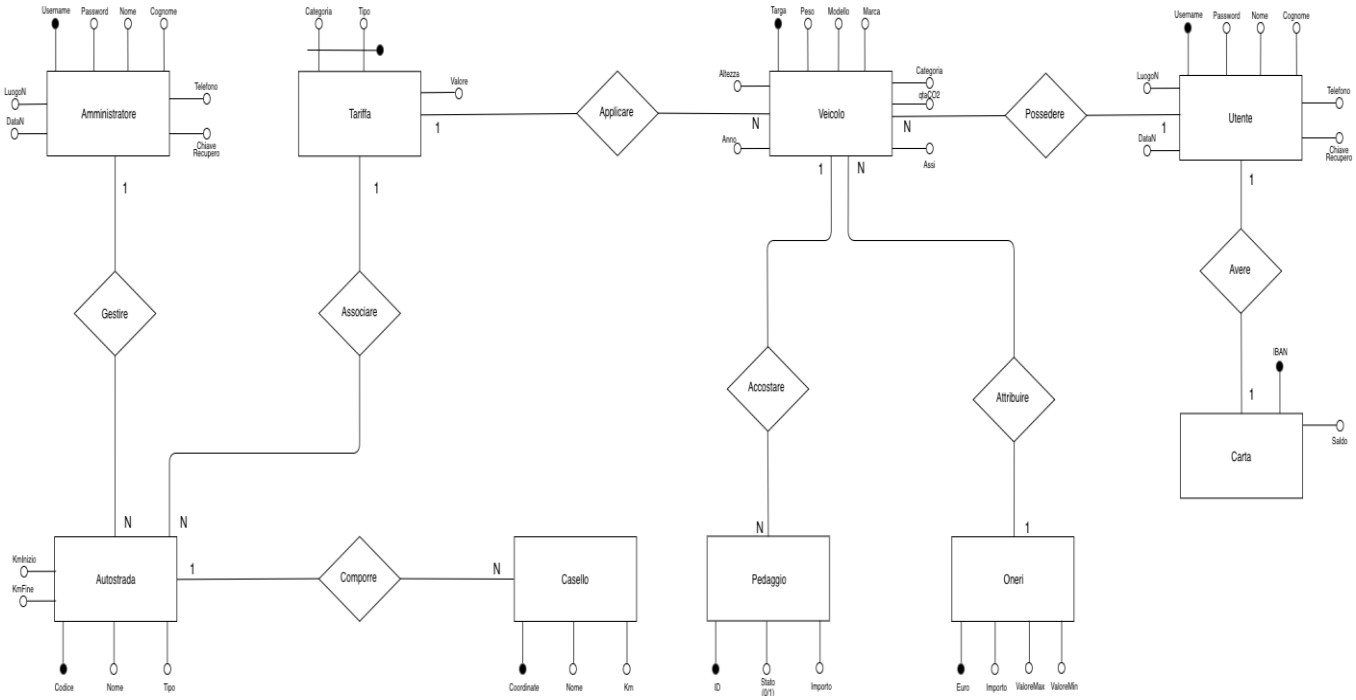
(2.3 Descrizione delle scelte e strategie adottate compresi Pattern Architetturali/Design)

Il team ha deciso di strutturare il package *"View"* in due subpackages chiamati rispettivamente *"DashboardAdmin"* e *"DashboardUtente"* in modo tale da distinguere tutte le componenti che riguardano la figura Amministratore dalla figura Utente. Le componenti *"Registrazione"*, *"Login"*, *"RecuperoCredenziali"* non sono situate all'interno dei due subpackages in quanto sono componenti che offrono servizi ad entrambe le figure di utenza che usufruiranno del sistema.

Per quanto riguarda il package "Controller" abbiamo deciso di strutturarlo con tre componenti, che sono: *"LoginController"*, *"GestoreUtenteController"*, *"GestoreAdminController"* in quanto direttamente collegate alle componenti dei subpackeges rispettivamente definiti nel package *"View"* discusso in precedenza. Nel dettaglio, la *"LoginController"* gestirà le interazioni tra il *"Model"* e le componenti inerenti alla "Login", *"GestoreUtenteController"* gestirà le interazioni tra il *"Model"* e le componenti inerenti al subpackages *"DashboardUtente"* ed infine *"GestoreAdminController"* gestirà le interazioni tra il "Model" e le componenti inerenti al subpackages *"DashboardAdmin".* Questa strutturazione definita in maniera sistematica renderà innanzitutto l'architettura del sistema più leggibile e quindi facilmente modificabile e inoltre garantirà l'aggiunta e la modifica di porzioni di progettazione senza impattare del tutto il sistema funzionante.

Infine il package *"Model"* è strutturato in due subpackages denominati *"Interfaces"* e *"Components"* in modo tale da distinguere tutte le componenti che riguardano le interfacce rispetto alle componenti che riguardano le loro implementazioni.

MODELLO E-R



MODELLO RELAZIONALE

Amministratore (Username, Password, ChiaveRecupero, Nome, Cognome, LuogoN, DataN, Telefono)

Autostrada (Codice, Nome, Tipo, KmInizio, KmFine, *Amministratore*)

Casello (Coordinate, Nome, Km, *Autostrada*)

Tariffa (Categoria, Tipo, Valore)

Veicolo (Targa, Marca, Modello, Peso, Assi, Altezza, Anno, Categoria, qtaCO2, *Oneri*, *Utente*)

Pedaggio (ID, Stato, Importo, *Veicolo*)

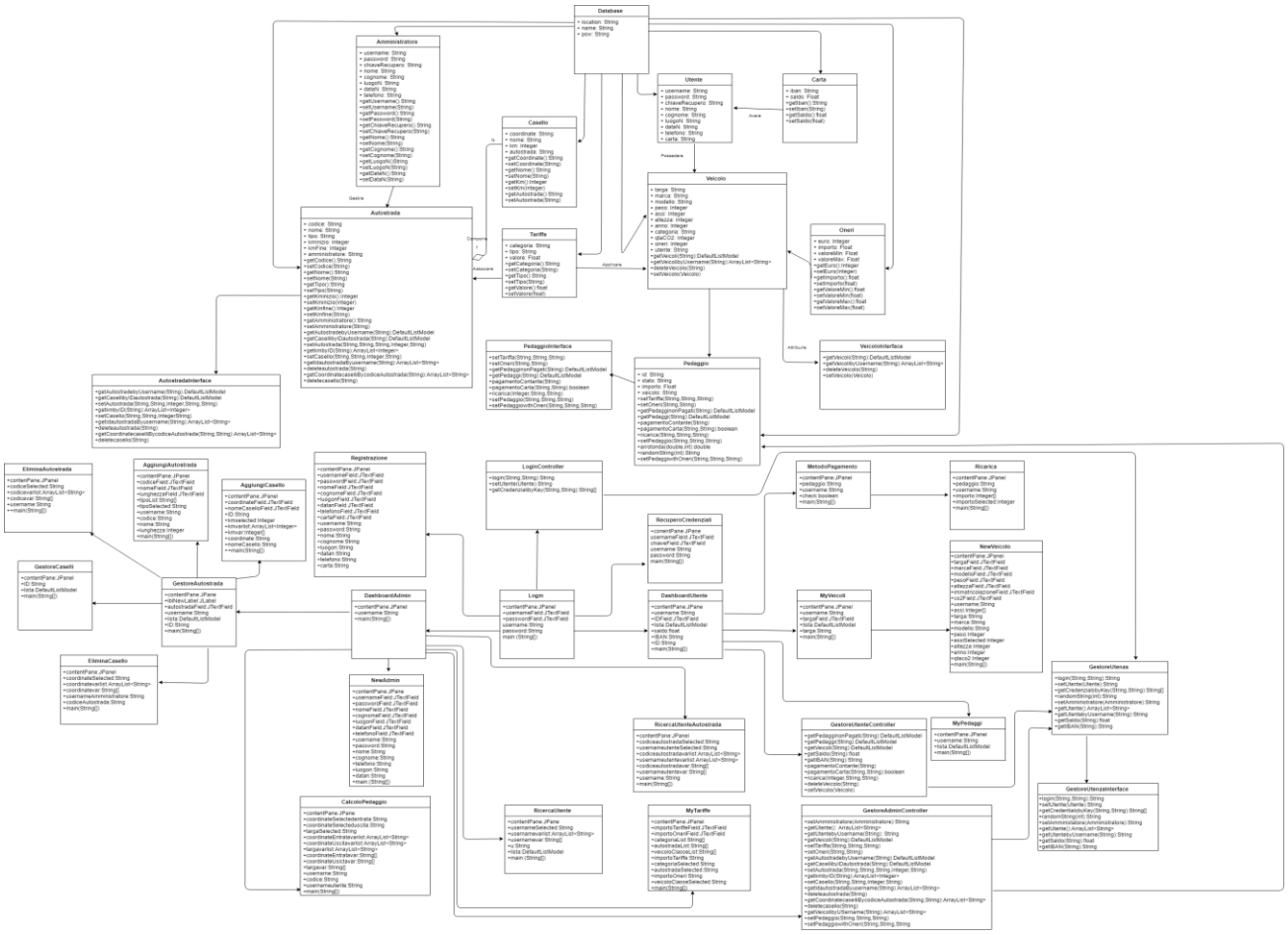
Oneri (Euro, Importo, ValoreMin, ValoreMax)

Utente (Username, Password, ChiaveRecupero, Nome, Cognome, LuogoN, DataN, Telefono, *Carta*)

Carta (IBAN, Saldo)

c. PROGETTAZIONE DI CLASSI E INTERFACCE

(3.1 Descrizione di classi, interfacce e membri)



Il **Class Diagram** riporta tutte le classi usate all'interno del nostro sistema software:

1. **DATABASE**: Rappresenta la classe situata nel file database.java presente nel package model.db che permette lo storage dei dati, infatti sono presenti metodi come Connect() con l'obiettivo di instaurare una connessione tra il database e il programma in java.
2. **AMMINISTRATORE**: E' una classe presente nel file Amministratore.java nel package model.components che permette di instanziare oggetti di tipo amministratore.
3. **CASELLO**: E' una classe presente nel file Casello.java nel package model.components che permette di istanziare oggetti di tipo casello. L'insieme degli oggetti di tipo casello rappresentano un agglomerato dell'oggetto di tipo Autostrada.
4. **UTENTE**: E' una classe presente nel file Utente.java nel package model.components che permette di istanziare oggetti di tipo utente.
5. **CARTA**: E' una classe presente nel file Carta.java nel package model.components che permette di istanziare oggetti di tipo carta.
6. **AUTOSTRADA**: E' una classe presente nel file Autostrada.java che implementa l'interfaccia AutostradaInterface. Questa classe è situata nel package model.components e permette di istanziare oggetti di tipo Autostrada.

Nel dettaglio, sono presenti le definizioni dei metodi: *"getAutostradebyUsername", "getCasellibyIDautostrada", "setAutostrada", "getkmbyID", "setCasello", "getIdautostradaByusername", "deleteautostrada", "getCoordinatecaselliBycodiceAutostrada", "deletecasello";*

7. **TARIFFA:** E' una classe presente nel file Tariffa.java situata nel package model.components che permette di istanziare oggetti di tipo Tariffa.
8. **VEICOLO**: E' una classe presente nel file Veicolo.java situata nel package model.components che permette di istanziare oggetti di tipo Veicolo.
   Nel dettaglio, sono presenti le definizioni dei metodi: "getVeicoli", "getVeicolibyUsername", "deleteVeicolo", "setVeicolo";
9. **ONERI**: E' una classe presente nel file Oneri.java situata nel package model.components che permette di istanziare oggetti di tipo Oneri.
10. **PEDAGGIO**: E' una classe presente nel file Pedaggio.java situata nel package model.components che permette di istanziare oggetti di tipo Pedaggio.
    Nel dettaglio, sono presenti le definizioni dei metodi: *"setTariffa", "setOneri", "getPedagginonPagati", "getPedaggi", "pagamentoContante", "pagamentoCarta", "ricarica", "setPedaggio", "arrotonda", "randomString", "setPedaggiwithOneri";*
11. **AUTOSTRADAINTERFACE**: E' un'interfaccia che verrà implementata dalla classe Autostrada, dove sono presenti le dichiarazioni dei metodi;
12. **PEDAGGIOINTERFACE**: E' un'interfaccia che verrà implementata dalla classe Pedaggio, dove sono presenti le dichiarazioni dei metodi;
13. **VEICOLOINTERFACE**: E' un'interfaccia che verrà implementata dalla classe Veicolo, dove sono presenti le dichiarazioni dei metodi;
14. **LOGINCONTROLLER**: E' una classe presente nel file LoginController.java situato nel package controller che permette di interfacciare le chiamate dei metodi effettuate dalla view mediante, nello specifico, l'utilizzo della chiamata al metodo corrispondente e che sarà presente nel model.components.
15. **GESTOREADMINCONTROLLER**: E' una classe presente nel file GestoreAdminController.java situato nel package controller che permette di interfacciare le chiamate dei metodi effettuate dalla view.dashboardadmin mediante, nello specifico, l'utilizzo della chiamata al metodo corrispondente e che sarà presente nel model.components.
16. **GESTOREUTENTECONTROLLER:** E' una classe presente nel file GestoreUtenteController.java situato nel package controller che permette di interfacciare le chiamate dei metodi effettuate dalla view.dashboardutente mediante, nello specifico, l'utilizzo della chiamata al metodo corrispondente e che sarà presente nel model.components.
17. **LOGIN:** E' una classe che implementa la classe JFrame. La classe Login è presente nel file Login.java nel package view che permette l'accesso alle dashboard da parte di un utente generico che utilizza il nostro sistema o da parte di un admin. E' una classe importante perché permette di effettuare il controllo sul tipo di Utente che sta provando ad accedere al sistema.
18. **REGISTRAZIONE**: E' una classe che implementa la classe JFrame. La classe Registrazione è presente nel file Registrazione.java nel package view che permette la registrazione da parte di un generico utente che utilizza il sistema software.

19. **RECUPEROCREDENZIALI**: E' una classe che implementa la classe JFrame. La classe RecuperoCredenziali è presente nel file RecuperoCredenziali.java nel package view che permette il recupero delle credenziali (username,password) da parte di un utente o di un admin mediante l'utilizzo di una chiave di recupero.
20. **DASHBOARDADMIN**: E' una classe che implementa la classe JFrame. La classe DashboardAdmin è presente nel file DashboardAdmin.java nel package view.dashboardadmin che permette all'admin di aggiungere, modificare ed eliminare un'autostrada con i rispettivi caselli, di gestire le tariffe unitarie e gli oneri per i costi esterni, di visualizzare le informazioni relative ad un determinato utente e i veicoli d'appartenenza e di effettuare il calcolo del pedaggio autostradale utilizzando le informazioni relative al percorso e alla targa del veicolo dell'utente. Inoltre, può aggiungere un nuovo admin all'interno del sistema.
21. **NEWADMIN**: E' una classe che implementa la classe JFrame. La classe NewAdmin è presente nel file NewAdmin.java nel package view.dashboardadmin che permette all'admin di aggiungere un nuovo amministratore. Tale classe permette la creazione di un oggetto di tipo Amministratore e una volta creato, l'oggetto verrà passato come parametro al gestoreadmincontroller.
22. **CALCOLOPEDAGGIO**: E' una classe che implementa la classe JFrame. La classe CalcoloPedaggio è presente nel file CalcoloPedaggio.java nel package view.dashboardadmin che permette all'admin di effettuare il calcolo del pedaggio utilizzando le informazioni relative al percorso e alla targa del veicolo dell'utente. Successivamente, la richiesta di pagamento del pedaggio verrà inviata all'utente di riferimento.
23. **GESTOREAUTOSTRADA**: E' una classe che implementa la classe JFrame. La classe GestoreAutostrada è presente nel file GestoreAutostrada.java nel package view.dashboardadmin che permette all'admin di aggiungere, modificare ed eliminare un'autostrada con i rispettivi caselli.
24. **RICERCAUTENTE**: E' una classe che implementa la classe JFrame. La classe RicercaUtente è presente nel file RicercaUtente.java nel package view.dashboardadmin che permette all'admin di visualizzare le informazioni relative ad un determinato utente e i veicoli d'appartenenza.
25. **RICERCAUTENTEAUTOSTRADA:** E' una classe che implementa la classe JFrame. La classe RicercaUtenteAutostrada è presente nel file RicercaUtenteAutostrada.java nel package view.dashboardadmin che permette all'admin di selezionare l'utente e l'autostrada inerente al pedaggio che si vuole realizzare.
26. **MYTARIFFE**: E' una classe che implementa la classe JFrame. La classe MyTariffe è presente nel file Mytariffe.java nel package view.dashboardadmin che permette all'admin di gestire/modificare le tariffe unitarie e gli oneri per i costi esterni inerenti ai veicoli e alla tipologia di autostrada.
27. **DASHBOARDUTENTE**: E' una classe che implementa la classe JFrame. La classe DashboardUtente è presente nel file DashboardUtente.java nel package view.dashboardutente che permette all'utente di aggiungere ed eliminare uno o più veicoli, visualizzare i pedaggi pagati e non pagati, di procedere al pagamento di un pedaggio mediante l'utilizzo di contanti o carta e di effettuare una ricarica sulla carta e visualizzare il saldo.

28. **MYPEDAGGI**: E' una classe che implementa la classe JFrame. La classe MyPedaggi è presente nel file MyPedaggi.java nel package view.dashboardutente che permette all'utente di visualizzare tutti i pedaggi ad esso corrispondenti.
29. **MYVEICOLI**: E' una classe che implementa la classe JFrame. La classe MyVeicoli è presente nel file MyVeicoli.java nel package view.dashboardutente che permette all'utente di visualizzare tutti i veicoli di sua appartenenza e di passare all'aggiunta o alla eliminazione di una di essi.
30. **ELIMINAAUTOSTRADA**: E' una classe che implementa la classe JFrame. La classe EliminaAutostrada è presente nel file EliminaAutostrada.java nel package view.dashboardadmin che permette all'amministratore di eliminare un'autostrada passando il codice;
31. **AGGIUNGIAUTOSTRADA:** E' una classe che implementa la classe JFrame. La classe AggiungiAutostrada è presente nel file AggiungiAutostrada.java nel package view.dashboardadmin che permette all'amministratore di aggiungere un'autostrada che sarà di sua competenza.
32. **AGGIUNGICASELLO:** E' una classe che implementa la classe JFrame. La classe AggiungiCasello è presente nel file AggiungiCasello.java nel package view.dashboardadmin che permette all'amministratore di aggiungere un casello che apparterrà al codice dell'autostrada.
33. **ELIMINACASELLO:** E' una classe che implementa la classe JFrame. La classe EliminaCasello è presente nel file EliminaCasello.java nel package view.dashboardadmin che permette all'amministratore di eliminare un casello che apparterrà al codice dell'autostrada.
34. **GESTORECASELLI:** E' una classe che implementa la classe JFrame. La classe GestoreCaselli è presente nel file GestoreCaselli.java nel package view.dashboardadmin che permette all'amministratore di visualizzare la lista dei caselli inerenti al codice dell'autostrada.
35. **NEWVEICOLO**: E' una classe che implementa la classe JFrame. La classe NewVeicolo è presente nel file NewVeicolo.java nel package view.dashboardutente che permette all'utente di aggiungere un nuovo veicolo.
36. **METODOPAGAMENTO**: E' una classe che implementa la classe JFrame. La classe MetodoPagamento è presente nel file MetodoPagamento.java nel package view.dashboardutente che permette all'utente di procedere al pagamento di un pedaggio mediante contanti o carta.
37. **RICARICA**: E' una classe che implementa la classe JFrame. La classe Ricarica è presente nel file Ricarica.java nel package view.dashboardutente che permette all'utente di effettuare una ricarica sulla carta nel caso in cui il saldo risulti minore dell'importo del pedaggio da pagare.
38. **GESTOREUTENZA**: E'una classe che implementa la classe GestoreUtenzaInterface dove sono presenti le definizioni dei metodi *"login", "setUtente", "getCredentialsbyKey", "randomString", "setAmministratore", "getUtente", "getUtentebyUsername", "getSaldo", "getIBAN"*. E' una classe molto importante dove è presente la logica della login e della modifica, restituzione e aggiunta dei dati inerenti all'amministratore e all'utente generico che utilizza il sistema. Inoltre, presenta anche la logica per il recupero delle credenziali d'accesso mediante il metodo getAdminbyKey(String) dove verrà passato come parametro la chiave di recupero.

39. **GESTOREUTENZAINTERFACE**: E’ un’interfaccia che verrà implementata dalla classe GestoreUtenza dove sono presenti le dichiarazioni dei metodi di get e set inerenti all’admin e all’utente generico che utilizza il sistema.

(3.2 descrizione dei dettagli di design scelti)

Il team ha deciso di strutturare il Class Diagram sulla base del Component Diagram. Nello specifico, i membri inerenti alle classi del package model.components rispettano gli attributi che sono stati definiti all’interno del modello relazionale. E’ stato inoltre definito il vincolo relativo all’autostrada come agglomerato di caselli in modo tale che, al momento della cancellazione di una determinata autostrada, verranno rimossi anche i caselli ad esso corrispondenti in modo tale da risparmiare le risorse in termini di occupazione di memoria.